PETUNIA PLAYER USER MANUAL

HUH ELECTRONICS

1429 Maple St.,
San Mateo, CA
94402
(415) 573-7359

THE PETUNIA PLAYER

User's Manual

by

Martin J. Cohen

October 18, 1978

COPYRIGHT 1978 BY HUH ELECTRONICS

Table of Contents

1 Introduction.....1

2 How to run the Petunia Player.....1
2.1 Loading the machine language programs.....1
2.2 Loading and running the BASIC program.....1

3 The Composition of the Petunia Player.....2
3.1 The BASIC subroutines.....2
3.2 Input to the BASIC subroutines.....3
3.2.1 Commands.....3
3.2.1.1 Begin and end chords.....3
3.2.1.2 Change the keyboard.....3
3.2.1.3 Setting the tempo.....4
3.2.1.4 Setting the duration of notes.....4
3.2.2 Notes.....5
3.2.2.1 The 4-octave keyboard.....5
3.2.2.2 The 2-octave keyboard.....5
3.2.3 Phrases.....6
3.2.3.1 Defining a phrase.....6
3.2.3.2 Playing a phrase.....7
3.2.3.3 Restrictions and comments.....7
3.2.4 Playing music.....7
3.2.4.1 Initializing the Petunia Player.....7
3.2.4.2 Playing stored music - what to respond to requests.....8
3.2.4.3 Special variables used in the BASIC routines.....8
3.2.4.3.1 ID, IE, IG, IH, and II.....8
3.2.4.3.2 C$, C.....9
3.2.4.3.3 S$(*), NS, LS, JS, and IS.....9
3.2.4.4 Playing music from the keyboard.....9
3.2.4.5 Playing music from DATA statements.....10
3.2.4.5.1 Format of the DATA statements.....10
3.2.4.5.2 Comments on the program listing of the tunes.....10
3.2.4.5.3 Lines 200-219.....10
3.2.4.5.4 Lines 230-315.....11
3.2.4.5.5 Lines 320-328.....11
3.2.4.5.6 Lines 340-345.....11
3.2.4.5.7 Lines 350-355.....11
3.2.4.5.8 Lines 400-598.....11
3.2.4.5.9 Lines 600-699.....11
3.2.4.5.10 Lines 1200-1299.....11
3.2.4.5.11 Lines 1600-1699.....11
3.2.4.5.12 Lines 2000-2099.....11
3.2.4.5.13 Lines 2100-2199.....12
3.2.4.5.14 Lines 2200-2300.....12

4 The machine language programs.....12
4.1 The Petunia Player loader.....12
4.1.1 Special input values.....12
4.1.2 Specifying where the routines and tables are loaded.....13
4.1.3 Numeric values.....13
4.1.4 Generating relocatable code.....13
4.2 What the loader loads.....13
4.3 After the loading is over.....14
4.4 Listing of PETUNIA#5.....14

Appendix

Listing of PETUNIA PLAYER.....Page 19
Listing of PETUNIA#5.....Page 24

Petunia Player's User Manual

1 Introduction

The Petunia Player, when used in conjunction with HUH Electronics' Petunia, allows music to be played on the PET either directly from the keyboard or stored and played as a series of notes. When music is played directly from the keyboard, the keyboard is treated as banks of keys, each bank slightly longer than one octave. When music is stored and then played as a series of notes, the notes are specified exactly as they would be entered from the keyboard. Thus, once a Petunia Player user knows to play music from the keyboard, he/she can easily store music (as DATA statements) that can be played back at any time.

2 How to run the Petunia Player

The purpose of this section is to let you get the Petunia Player loaded and running. More detailed descriptions of the components of the Petunia Player are in the following sections, and are referenced here as needed.

The Petunia Player is divided into two parts. The first is a set of machine language routines which do that part of the music making which BASIC is too slow to do. The second is built around some BASIC subroutines which use the machine language routines to make music.

2.1 Loading the machine language programs

Load the program which contains the machine language component of the Petunia Player. This is currently called PETUNIA#5. Once this program is loaded, if your PET is a standard 8K model, just enter RUN (3 letters, not the shifted-STOP). If your PET has more memory, change line 15 so that the part containing 'LH=28:' is changed to 'LH=4*N-4:', where N is the number of K in your machine (for example, if you a fortunate enough to have a 32K PET, change the line to 'LH=124:'). Then enter RUN. After a series of printouts on the screen describing the location of various items, the message

PETUNIA SUBROUTINE IS NOW LOADED
AND PROTECTED

will be displayed, followed by READY. The machine language programs are now in memory, and you can proceed to the next step.

2.2 Loading and running the BASIC program

The second program making up the Petunia Player uses the machine language routines to play music either directly from the PET's keyboard or from a built in (and easily changable) repertoire of tunes. This program is currently called PLAYER#5. Load this program into the PET; it will coexist with the machine language routines previously loaded.

To play music directly from the keyboard, enter 'RUN1ØØ'. You are then able to play notes by pressing any of the keys. The meanings of the keys is described in the section titled 'Input to the BASIC subroutines' and, in particular, in the subsection titled 'The 4-octave keyboard'. By using the '@' key, you can switch to the 2-octave keyboard, which is described in the subsequent section.

To choose from a selection of presupplied tunes, just enter 'RUN' (again, 3 letters, not shifted-STOP). The subsequent running is supposed to be self-explanatory. Our only comments for now are that

(1) to play the tunes in the proper tempo, you should respond '2' to the message

ENTER 1 TO PLAY AS IS, 2 TO STORE FIRST

and (2) when the message

PRESS ANY KEY TO PLAY

is displayed, do not press the STOP key unless you want to stop the program.

3 The Composition of the Petunia Player

Since the BASIC routines are the ones which the user will directly interface with, they will be described first. The machine language routines and tables are then briefly described - a fuller description will be included in the final version of this documentation.

Listings of all programs which make up the Petunia Player and the assembly language versions of the machine language routines are included.

3.1 The BASIC subroutines

There are two BASIC subroutines available. This reflects the two ways in which music can be played. The first routine plays one note (or chord) at a time. Each time it is called, it plays the note specified (unless the input to it indicates that a non-playing parameter has been passed, such as 'start a chord' or 'set the duration of subsequent notes'). This is used when you would want to mess around at the keyboard or want to play a series of notes but want to be able to interrupt them (for example, when checking out some music).

The second routine is called in exactly the same manner as the first. However, instead of immediately playing the note specified, this routine stores an encoded version of the information passed to it in memory. When all the desired musical information has been stored, an auxilliary routine is called which causes the whole

series of notes to be sequentially played. This routine can be used after the music has been debugged. It has the great advantage that all music played by it (including chords) is played in the proper tempo and duration. The first routine, which plays each note as soon as it can, is slowed up by the necessity for BASIC to interpret the input - this causes noticable delays when music is played (especially chords). The disadvantage of the second routine is that once the music has started to play, there is absolutely no way to stop it until the music finishes - the machine language routine it calls completely ignores any outside activity.

The first subroutine is called by a GOSUB 4ØØ, and the second is called by a GOSUB 6ØØ. Further details about these routines are given later on.

3.2 Input to the BASIC subroutines

All input to the BASIC subroutines at statements 4ØØ and 6ØØ are obtained through the subroutines at statements 2ØØ through 217. The routine entered at 2ØØ gets the next input character and places it in the character variable C$. If the variable II is 2, the character is gotten from the keyboard; otherwise the character is gotten from the array of string variables S$(Ø to NS). The routine entered at 21Ø both gets the next character into C$ and decomposes it into C; if this routine is entered at 217, it decomposes the previous contents of C$.

The input is divided into three classes - commands, notes, and phrases.

3.2.1 Commands

3.2.1.1 Begin and end chords

The character '/' (slash) both begins and ends a chord. If a chord is not being entered and a '/' is found, the following notes are setup so that they will be played simultaneously. The end of the chord is indicated by another '/'.

If more than 4 notes are specified for the chord, only the first three and the last will be played.

3.2.1.2 Change the keyboard

The standard keyboard is the 4-octave version, in which 4 octaves of 'white notes' can be directly played and the 'black notes' are played by using the 'shift' key (the format of the keyboards is described in detail later). This can be changed

to a 2-octave version, in which 2 octaves of both 'black' and 'white' notes can be directly played and the other 2 octaves accessed via the 'shift' key by using the character '@'. When an '@' is found, the board mode is switched - if the current keyboard is 4 octave, it becomes 2 octave, and if it is currently 2 octave, it becomes 4 octave.

Note: when music is being played directly from the keyboard (i.e, when variable ID is 1), the board which then comes into use is identified by the display '<2>' for the 2-octave keyboard and '<4>' for the 4-octave keyboard.

3.2.1.3 Setting the tempo

The tempo can vary from Ø (zero) through 255. The value desired is passed to the Petunia machine language routine by POKEing at a certain location. To make setting the tempo easier, when the character '=' is found, the following character specifies the tempo. The tempo is computed as follows: A = 1Ø, B = 2Ø, C = 3Ø, ..., Y = 25Ø. If the letter is shifted, 5 is added to the tempo value, so we get a = 15, b = 25, c = 35, .., y = 255. No other characters should be used to specify a tempo except letters and shifted letters, or funny things may happen.

Note: the greater the tempo, the slower the music will be played.

3.2.1.4 Setting the duration of notes

The tempo applies to all music played. It is essentially a repetition factor specifying how many times a particular inner loop (which generates the note) is executed. This inner loop is contained within a (what else?) outer loop, and the number of times this outer loop is executed is set by the duration value.

The BASIC routines used here use the convention that the digit keys (both unshifted and shifted) specify the duration of notes. Once a duration has been specified, all notes are played using that duration until a different duration happens along. The correspondence between key and length of note is Ø = whole, 1 = half, 2 = quarter, 3 = eighth, 4 = sixteenth, 5 = thirty-second, 6 = sixty-fourth, 7 = triplet half, 8 = triplet quarter, and 9 = triplet eighth. If the keys Ø through 6 are shifted, the duration of the notes they specify is increased by 5Ø% (i.e., the note is 'dotted'). A triplet note is one-third the duration of the note it is a triplet of, so that 3 triplet half notes have the same duration as a standard half note.

3.2.2 Notes

Notes are entered by considering the keyboard to represent the banks of an organ or piano (if such a piano exists). There are 2 versions of the keyboard available.

3.2.2.1 The 4-octave keyboard

In this keyboard, the keys represent the 'white keys' on a standard keyboard. The following shows which key represents which note. The numbers following the note (A through G) represent the octave number, with octave 1 being lowest.

```
  !   "   #   $   %   '   &   \   (   )
 B3  C4  D4  E4  F4  G4  A4  B4  C5

  Q   W   E   R   T   Y   U   I   O   P
 B2  C3  D3  E3  F3  G3  A3  B3  C4  D4

  A   S   D   F   G   H   J   K   L   :
 B1  C2  D2  E2  F2  G2  A2  B2  C3  D3

  Z   X   C   V   B   N   M   ,   ;   ?
     C1  D1  E1  F1  G1  A1  B1  C2  D2
```

Thus, to play the lowest C, X should be used, and to play the scale of the second highest octave, the keys W, E, R, T, Y, U, I, O should be sequentially used.

To play the 'black keys' on the 4-octave keyboard, use the 'shift' key. When a shifted key is found, the note one half-tone higher than that indicated by the unshifted key is played. For example, to play the C# in octave 2, use shifted-S.

This applies to all the keys, not just those which, on a piano or organ keyboard, have black keys. For example, shifted-V would play F1, the same as unshifted-B.

3.2.2.2 The 2-octave keyboard

As its name implies, the 2-octave keyboard makes only 2 octaves available without using the shift key. However, it makes up for this by making all the notes of the chromatic scale available for those 2 octaves. The meaning of the keys in the 2-octave keyboard is

!	"	#	$	%	'	&	\	(	)
C4	C#4	D#4	F4	F#$	G#4	A#$	C5		
Q	W	E	R	T	Y	U	I	O	P
B3	C4	D4	E4	F4	G4	A4	B4	C5	
A	S	D	F	G	H	J	K	L	:
C3	C#3	D#3	F3	F#3	G#3	A#3	C4	C#4	D#4
Z	X	C	V	B	N	M	,	;	?
B2	C3	D3	E3	F3	G3	A3	B3	C4	D4

To obtain the lower 2 octaves, use the shift key. When a shifted key is used, the note 2 octaves lower than that represented by the unshifted key is played. For example, to play the lowest C, use shift-X.

3.2.3 Phrases

A phrase is a set of commands and/or notes which can be referred to as a single entity. Once a phrase has been defined, it can then be played back any number of times merely by referring to it. Phrases can be used in any situation where the same music is to be played more than once, with the most common usage being in repeats.

3.2.3.1 Defining a phrase

There can be up to ten phrases in existence at any time, each identified by a digit from 0 to 9. The character which identifies the start of a phrase is the left bracket ('['). This is immediately followed by the digit which identifies the phrase. For example, to define phrase 3, begin with '[3'.

Following the start of the phrase comes the body of the phrase. This is any combination of commands and/or notes. Note that, inside a phrase, another phrase can neither be defined nor referred to. The phrase is ended by the right bracket (']').

For example, to define a scale as phrase 2, you could do

[2QWERTYUI]

3.2.3.2 Playing a phrase

To play a phrase, just use the greater than ('>') followed by the digit identifying the phrase to be played. For example, to play phrase 2, just do '>2'. If phrase 2 is the scale above, to play it first as whole notes, then as half notes, quarter notes, .., 64-th notes, just do 'Ø>21>22>23>24>25>26>2'.

3.2.3.3 Restrictions and comments

1. As stated above, there can be at most ten phrases available at any one time.

2. A phrase can neither be defined nor played from inside another phrase.

3. The most recent definition of a phrase is the one in use at any time.

3.2.4 Playing music

There are 2 ways to play music with the Petunia Player. It can be played directly from the keyboard, or the music can be stored in an array of string variables and played sequentially by a program.

3.2.4.1 Initializing the Petunia Player

No matter how the music is played, some initializing code must be executed. The purpose of this code is to setup the PET for outputting (the default for the parallel port is to input), and compute various addresses and a table needed by the 2 BASIC subroutines. To allow for the Petunia Player's machine language routines to be anywhere in memory, the variable LH is set to the number of the first page of the 65Ø2's memory used by these routines. The standard value is LH=28, which corresponds to a PET with 8K of memory. If your PET has N K of memory, set LH=4*N-4. This allows 1K for the machine language routines and their tables. For example, in a 32K PET (nice work if you can get it!), set LH=124.

In the Petunia Player, this code is a subroutine which begins at line 12ØØ. To initialize, after the code setting LH has been placed in the program (make sure it is there, and not just executed directly), have your program do a GOSUB 12ØØ. The PET will now be ready to play.

3.2.4.2 Playing stored music - what to respond to requests

When you do a 'RUN' to play music, you are asked

ENTER 1 TO PLAY AS IS, 2 STORE FIRST

As described above, if you enter '2', the music will be processed and placed in the video memory for PETUNIA#5 to play. If you enter '1', the music will be played as soon as the information is read. This has the disadvantage that the tempo will not be right due to the time taken by BASIC to process the music description. However, the music can be interrupted at any time, which makes it much easier to correct errors in the music. To further aid in the correction of mistakes, two more items are asked for in this mode.

First, you are asked

ENTER 1 TO DISPLAY NOTE INPUT, 2 NOT TO

If you reply '1', as each character is processed, it is displayed on the screen. This includes characters in phrases, both when being initially scanned and when being played.

Second, you are asked

ENTER 1 FOR CONTINUOUS PLAY, 2 FOR DEBUG

If you enter '1', the notes will be played continuously, with only 'STOP' being able to interrupt then. If you enter '2', you will be in debug mode. In this mode, each time you press a key, a single note or chord is played. Thus, individual notes can be examined. To make this more convenient, if the '=' is pressed, PLAYER#5 then enters continuous play mode, and remains there until '=' is pressed again, when debug mode it again entered. This allows you to go quickly through correct parts of your music, and single-step through parts you want to change.

3.2.4.3 Special variables used in the BASIC routines

3.2.4.3.1 ID, IE, IG, IH, and II

These variables control the type of actions taken during execution. They are essentially flags, and have the following meanings:

ID - if 1, display the notes on the terminal as they are played; otherwise do not.

IE - if 1, play the notes immediately; if 2, store the notes for later play at the proper tempo

IG - if 1, play music continuously; if 2 and IH=1, wait for a key to be pressed before playing each note (this is the 'debug' mode of play).

IH - see IG, above.

II - if 1, get the next character from the array of string variables S$(*); otherwise get it from the keyboard.

3.2.4.3.2 C$, C%, and C1%

These variables contain the character to be interpreted by the routines entered at 400 and 600. They are gotten by the routines at statements 200 through 219.

C$ is the character itself. If it is null, the music is at its end.

C% is C$ converted to numeric. It is gotten by 'C% = ASC(C$)'.

C1% is the low order 7 bits of C%. It is gotten by 'C1% = C% AND 127'.

3.2.4.3.3 S$(*), NS, LS, JS, and IS

These variables contain the music to be played and the current position in it. They are used in the code in statements 200 to 219.

The mixture of commands, notes, and phrases is stored in S$(j), where j goes from 0 to NS. The current value of j is in JS, the length of S$(JS) is in LS, and the position in S$(JS) of the character currently in use in C$ is in IS. We thus always have 0 <=JS <=NS and 1 <=IS <=LS.

3.2.4.4 Playing music from the keyboard

To play music from the keyboard, all that has to be done is to initialize ID through II properly (in particular, we must set II=2 to get the input from the keyboard) and do a 'GOSUB 400'. This is done in lines 100-130 of PLAYER#5.

3.2.4.5 Playing music from DATA statements

As explained above, the 2 BASIC subroutines get their input from the array of character variables S$, and do not care how S$ was gotten. Therefore, to play music from DATA statements, we need only decide how the statements are to be entered in the program, and write a small program which reads these DATA statements, stores them in S$(*), sets NS to the maximum index used, initialize IS, JS, and LS, and calls the desired BASIC subroutine. This is done in statements 21ØØ to 23ØØ in PLAYER#5.

Some sample music is supplied as data statements numbered from 5ØØØ upwards. The format of the data statements and the contents of the program will now be described.

3.2.4.5.1 Format of the DATA statements

Each song begins with the name of the song in quotes (for example, 'HAPPY BIRTHDAY'). Then comes the body of the tune. This is a series of DATA statements containing character values which are exactly those that would be used if the tune were to be played directly from the keyboard. The tune is ended by the characters "END". The last tune is followed by the character string "END OF TUNES".

3.2.4.5.2 Comments on the program listing of the tunes

Because of the non-standard character set used in the PET, in the attached listing of the program any shifted characters in the data statements making up the tune descriptions are printed as if they were not shifted. To show the true characters, the code from lines 4ØØØ through 4Ø9Ø reads every data statement and prints their contents in unshifted mode but with an up-arrow immediately preceding each shifted character. The output of this routine is provided following the listing of lines 5ØØØ and beyond.

3.2.4.5.3 Lines 2ØØ-219

Gets the next character (either from the keyboard or from the array S$(*)) and may also decompose it.

3.2.4.5.4 Lines 23Ø-315

Given a note or duration in C$, processes it. If the Petunia Player is in immediate mode (i.e., IE=1), the input is acted upon; otherwise, the appropriate conversion is done and the result stored in memory to be played later.

3.2.4.5.5 Lines 32Ø-328

Get the next character, and convert it to a tempo value.

3.2.4.5.6 Lines 34Ø-345

Switch between 2-octave keyboard and 4-octave keyboard.

3.2.4.5.7 Lines 35Ø-355

Get a character from the keyboard into C$.

3.2.4.5.8 Lines 4ØØ-598

Process commands, notes, or phrases in immediate mode (i.e., play the music as soon as possible).

3.2.4.5.9 Lines 6ØØ-699

Store commands, notes, and phrases for later playing by the code in statements 22ØØ-23ØØ.

3.2.4.5.1Ø Lines 12ØØ-1299

Initialize the Petunia Player.

3.2.4.5.11 Lines 16ØØ-1699

Either find a particular selection from tunes stored in DATA statements or list the numbers of all of them.

3.2.4.5.12 Lines 2ØØØ-2Ø99

Setup to play tunes - get all the necessary parameters so that the routines will run as desired.

3.2.4.5.13 Lines 2100-2199

After the code in lines 1600 to 1699 has positioned the DATA pointer properly, initialize the playing routines, read in the music, and either play it (if in immediate mode) or store it to be played.

3.2.4.5.14 Lines 2200-2300

Play a tune after it has been stored.

Note that once the music begins playing, it cannot be stopped until its end is reached.

4 The machine language programs

The machine language programs used by the Petunia Player consist of a set of routines and tables. The routines can be called either from BASIC or by each other, and the tables can be used similarly. We will first describe the loader used to load this stuff into memory, and then describe the tables and the routines.

4.1 The Petunia Player loader

The Petunia Player loader is designed to be able to load relocatable code. This code can start at any page in the 6502's meory, and can be specified in either decimal or hexadecimal for each byte.

4.1.1 Special input values

Some input values cause the loader to do special things. They are:

-1 - end loading of numbers.

-2 - display the location where the next byte will be loaded both in decimal from 0 through 65535 and as a page number and offset within the page. This information will be displayed together with the next input item on the screen of the PET.

-3 - switch the input mode. If it is currently decimal, change to hexadecimal; if currently hexadecimal, change to decimal.

Others could be added as needed. One which might be useful is to set the location counter in some way.

4.1.2 Specifying where the routines and tables are loaded

The code can start at any page number in the 65Ø2's memory. As in the initialization subroutine for the BASIC subroutines, the initial page number is in the variable LH and set in line 15. The default is LH=28 for a 8K PET. If your PET has more memory, set LH=4*N-4 for a N K PET (e.g., 124 for a 32K PET).

4.1.3 Numeric values

Numeric values are specified one value per byte. Values are initially assumed to be decimal, but the mode can be changed as described above to hexadecimal.

4.1.4 Generating relocatable code

A value greater than 255 can be generated in decimal mode just by entering the number or in hexadecimal mode by using GØ-GF for 256 through 272. If a value greater than 255 is found, 256 is subtracted from it and LH, the initial page number, is added to it. This effectively relocates the code.

4.2 What the loader loads

The loader loads the following items in the order specified.

The note table - this is the description of the actual note output by the music program.

The note program and its data area - this routine converts from a note number in the range Ø to 49 to a note value 2 bytes long as required by the music program.

The music program - this is the basic music generation program supplied by HUH Electronics.

The series of notes player - this routine is passed a page number by BASIC via the USR command and plays the notes specified at the memory locations beginning at that page.

Standard note table - this is used to convert a note number from Ø to 49 to a 2-byte note value. This can be used by both machine language and BASIC routines.

The keyboard table - after the loader loads all code specified, it then, in a section of code headed 'FILL THE NOTE TABLES', generates the tables needed to convert the 2octave and 4-octave keyboards to note numbers from Ø to 49.

The routine at statement 2ØØØ is a decimal and hex memory dump used

to debug the data entered.

4.3 After the loading is over

When loading is complete, the loader POKEs location 135 to the value in LH. This protects the machine language information so that when the BASIC part of Petunia Player is used, it will be safe.

4.4 Listing of PETUNIA#5

The following is a listing of PETUNIA#5. Because an assembler was not available to the author, the listing is hand entered (including the assembled part on the left!). Because of this, there may be some typos. However, the distributed version of PETUNIA#5 does work, and any conflicts are to be resolved in favor of the distributed version.

As described above, bytes of the form GØ through GF represent relocatable code, with GØ representing the starting page where the code goes.

```
                ;
                ; THE FOLLOWING CODE IS INSERTED BEFORE HUH'S PETUNIA
CODE
                ; (THAT CODE IS RELOCATED)
                ;
 G117 ØØ...     NOTE1   WORD Ø,Ø,Ø,Ø       STORAGE FOR NOTE NUMBERS
                ;
                ; PLAY NOTES USING THE NOTE TABLE
                ;
 G11F 18        NOTES   CLC                CLEAR CARRY FOR THE ROTATE
 G12Ø A2Ø8              LDX #8             4 NOTES = 8 BYTES
 G122 BD15G1    LOOP    LDA NOTE1-2,X      GET A NOTE NUMBER
 G125 2A                ROL A              TIMES 2 FOR 2-BYTE LOOKUP
 G126 A8                TAY
 G127 BDD1G2            LDA NOTBL+1,Y      GET A BYTE
 G12A 9DØAG1            STA TV1IN-2,X      STORE IN PETUNIA'S TABLE
 G12D B9DØG2            LDA NOTBL,Y        SAME FOR OTHER BYTE
 G13Ø 9DØBG1            STA TV1IN-1,X
 G133 A9ØØ              LDA #Ø             ZERO THE NOTE NUMBER
 G135 9D15G1            STA NOTE1-2,X
 G138 CA                DEX                SET FOR NEXT TIME
 G139 C1                DEX
 G13A DØE6              BNE LOOP           DO 4 TIMES
                ;
                ; HUH'S CODE FOLLOWS, STARTING AT G13C
                ;
                ; ***************************************************
********
                ;
```

```
                ; THE FOLLOWING CODE PLAYS A SERIES OF NOTES
                ;
                ; THE INITIAL LOCATION IS THE START OF THE PAGE PASSED
 BY BASIC
                ;
                ; IT RETURNS WHEN A VALUE GREATER THAN 25Ø IS FOUND
                ;
 G1AØ 2ØEØG1  SERIES  JSR INIT          SET THE NOTE POINTER
 G1A3 AØØØ            LOOP              LDY #Ø FIRST NOTE OF CHORD
 G1A5 2ØFØG1  LOOP1   JSR NEXT          GET THE NEXT INPUT
 G1A8 C9FA            CMP #25Ø          SEE IF WANT TO EXIT
 G1AA 9ØØ1            BCC *+3           NO
 G1AC 6Ø              RTS               YES - RETURN
 G1AD C9F9            CMP #249          SEE IF TEMPO
 G1AF DØØ9            BNE *+11          NO - SKIP THE CODE
 G1B1 2ØFØG1          JSR NEXT          GET THE NEXT BYTE
 G1B4 8D15G1          STA TEMPO         STORE IN HUH'S TEMPO PLACE
 G1B7 4CA5G1          JMP LOOP1         THEN GET THE NEXT NOTE
 G1BA 297F            AND #127          NOTE OR DATA - ISOLATE LOW B
ITS
 G1BC C94Ø            CMP #64           SEE IF DURATION
 G1BE 9ØØC            BCC *+14          NO - WILL BE NOTE
 G1CØ 2Ø3F            DURATION - GET POINTER
 G1C2 AA              TAX
 G1C3 BDØØG2          LDA DURTBL,X      GET DURATION VALUE FROM TABL
E
 G1C6 8D14G1          STA DUR           STORE IN HUH'S DURATION PLAC
E
 G1C9 4C A5G1         JMP LOOP1         GET THE NEXT THING
 G1CC 9917G1          STA NOTE1,Y       A NOTE - STORE THE NOTE NUMB
ER
 G1CF CØØ6            CPY #6            SEE IF STORING AS THE 4TH NO
TE
 G1D1 FØØ2            BEQ *+4           YES - DO NOT STORE BEYOND
 G1D3 C8              INY               NO - CAN STORE ANOTHER
 G1D4 C8              INY
 G1D5 2ØF8G1          JSR LOAD          RECALL THE INPUT
 G1D8 1ØCB            BPL LOOP1         IF POSITIVE, DO NOT PLAY - G
ET MORE
 G1DA 2Ø1FG1          JSR NOTES         NEGATIVE - PLAY IT
 G1DD 4CA3G1          JMP LOOP          THEN CONTINUE
                ;
                ; INITIALIZING CODE - SET START OF INPUT
                ;
 G1EØ 2ØA7DØ  INIT    JSR $DØA7         GET PAGE OF START OF MUSIC
 G1E3 A6B4            LDX $B4           THAT'S THE PAGE IN X NOW
 G1E5 CA              DEX               POINT TO THE PREVIOUS PAGE
 G1E6 8EFAG1          STX LOAD+2        STORE AS PAGE NUMBER
 G1E9 A9FF            LDA #255          POINT TO END OF THAT PAGE
 G1EB 8DF9G1          STA LOAD+1        STORE THAT LOC
 G1EE 6Ø              RTS                (POINTER IS NOW JUST BEFORE
 THE MUSIC)
                ;
```

```
                ; THIS ROUTINE GETS THE NEXT BYTE OF MUSIC
                ;
 G1FØ EEF9G1    NEXT     INC LOAD+1          INCREMENT LOW ORDER BYTE
 G1F3 DØØ3               BNE *+5             IF NON-ZERO, HIGH ORDER IS O
K
 G1F5 EEFAG1             INC LOAD+2          INCREMENT PAGE NUMBER
 G1F8 ADØØØØ    LOAD     LDA **              GET A BYTE (STORED ON)
 G1FB 6Ø                 RTS                 RETURN WITH IT
 G1FC ØØ....             BYTE Ø,Ø,Ø,Ø        FILLER
                ;
                ; THE DURATION TABLE
                ;
 G2ØØ VALUES    DURTBL   BYTE 128,64,32,16,8,4,2,1 STANDARD NOTES
                         BYTE 192,96,48,24,12,6,3,1 DOTTED NOTES
                         BYTE 43,21,11,5,3,1,1,Ø TRIPLETS
                ;
                ; ***************************************************
********
                ;
                ; THE FOLLOWING CODE IS THAT CALLED FROM BASIC
                ;
                ; IT IS PASSED THE STARTING PAGE NUMBER
                ;
                ; IT HANDLES MUSICAL PHRASES
                ;
                ; ALL OTHER STUFF IS DONE BY THE ROUTINE SERIES
                ;
                ; ***************************************************
**
                ;
                ; SPECIAL VALUES OF INPUTS:
                ;
                ; 255 - END OF DATA
                ; 254 - START A PHRASE
                ; 253 - END A PHRASE
                ; 252 - USE A PHRASE (NUMBER FOLLOWS)
                ; 249 - SET TEMPO (TEMPO FOLLOWS)
                ;
                ; START BEYOND THE END OF OTHER ROUTINES
                ;
 G22Ø           * = $G22Ø
 G22Ø 2ØEØG1    MUSIC    JSR INIT
 G223 A214               LDX #2Ø             CLEAR 1Ø POINTER WORDS (= 2Ø
BYTES)
 G225 A9ØØ               LDA #Ø
 G227 9D47G2    CLEAR    STA GRPLOC-1,X      ZERO THEM OUT
 G22A CA                 DEX
 G22B DØFA               BNE CLEAR
 G22D 2ØA3G1    PLAY     JSR LOOP            PLAY STANDARD UNTIL SPECIAL
 G23Ø C9FF               CMP #255            CHECK FOR END
 G232 DØØ1               BNE *+3             NOPE
 G234 6Ø                 RTS                 YEP - RETURN
 G235 C9FE               CMP #254            START OF GROUP?
```

```
 G237 FØ27              BEQ STRTGP         YES
 G239 C9FC              CMP #252           USE A PHRASE?
 G23B FØ43              BEQ USEGP          YES
 G23D C9FD              CMP #253           END PHRASE?
 G23F FØ67              BEQ ENDGRP         YES
 G241 422DG2            JMP PLAY           NONE OF ABOVE - JUST PLAY
 G248            * = *+4 FILLER
 G248 ØØØØØØ     GRPLOC WORD Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø  POINTERS TO PHRASES
 G26Ø            * = *+4                    FILLER
                 ;
                 ; START A PHRASE (OR GROUP)
                 ;
 G26Ø 2ØCØG2     STRTGP JSR GETGRP          GET THE NUMBER
 G263 3ØØD              BMI SG1            IF NEGATIVE, SKIP IT (BAD)
 G265 AA                TAX                SAVE FOR INDEXED USE
 G266 ADF9G1            LDA LOAD+1         SAVE LOC OF START OF PHRASE
 G269 9D48G2            STA GRPLOC,X
 G26C ADFAG1            LDA LOAD+2
 G26F 9D48G2            STA GRPLOC+1,X
 G272 2ØFØG1     SG1    JSR NEXT           SCAN FOR THE END OF THE PHRA
SE
 G275 C9FD              CMP #253           ... INDICATED BY 253
 G277 DØF9              BNE SG1
 G279 4C2DG2            JMP PLAY           THEN PLAY IT AGAIN
 G28Ø            * = *+4                    FILLER
                 ;
                 ; USE A PHRASE PREVIOUSLY ENCOUNTERED
                 ;
 G28Ø 2ØCØG2     USEGRP JSR GETGRP          GET THE PHRASE NUMBER
 G283 3ØA8              BMI PLAY           IF INVALID, IGNORE
 G285 AA                TAX                PLACE IN X FOR INDEXED ACCES
S
 G286 ADF9G1            LDA LOAD+1         SAVE WHERE CURRENTLY AT
 G289 8DB8G2            STA SAVE
 G28C ADFAG1            LDA LOAD+2
 G28F 8DB9G2            STA SAVE+1
 G292 BD49G2            LDA GRPLOC+1,X     GET MSB OF PHRASE LOCATION
 G295 FØØ9              BEQ *+11           IF NON-EXISTENT, DO NOT DO I
T
 G297 8DFAG1            STA LOAD+2         OK - STORE THE PHRASE LOCATI
ON
 G29A BD48G2            LDA GRPLOC,X       GET LSB
 G29D 8DF9G1            STA LOAD+1         STORE THAT
 G2AØ 4C2DG2            JMP PLAY           PLAY THE PHRASE
 G2A8            * = *+5                    FILLER
                 ;
                 ; THE END OF A PHRASE
                 ;
 G2A8 ADB8G2     ENDGRP LDA SAVE           GET WHERE SWITCHED FROM
 G2AB 8DF9G1            STA LOAD+1         SET AS WHERE TO START AGAIN
 G2AE ADB9G2            LDA SAVE+1
 G2B1 8DFAG1            STA LOAD+2
 G2B4 4C2DG2            JMP PLAY           PLAY AGAIN
```

```
G2B8          * = *+1                    FILLER
G2B8          SAVE    WORD Ø             WHERE PHRASE CALLED FROM
G2CØ          * = *+6                    FILLER
              ;
              ; GET A PHRASE NUMBER
              ;
G2CØ 2ØFØGA   GETGRP  JSR NEXT           GET THE NEXT BYTE
G2C3 1ØØ1             BPL *+3            MUST BE NON-NEGATIVE
G2C5 6Ø               RTS                ELSE, RETURN AS BAD
G2C6 C9ØA             CMP #1Ø            MUST BE < 1Ø
G2C8 9ØØ3             BCC *+5
G2CA A9FF             LDA #255           ELSE, RETURN A NEG VALUE AS
BAD
G2CC 6Ø               RTS
G2CD ØA               ASL                OK - DOUBLE TO POINT TO WORD
SIZED TABLE
G2CE 6Ø               RTS                RETURN
G2DØ          * = *+1                    FILLER TO MULT OF 16
```

Listing of PETUNIA PLAYER, Version 5

```
1 REM PETUNIA PLAYER, VERSION 5 (FOR USE WITH PETUNIA#5)
2 REM BY MARTIN J. COHEN
3 REM COPYRIGHT 1978 BY HUH ELECTRONICS
10 DIM S$(50)
15 GOSUB 1200:GOTO2000
100 REM PLAY NOTES
110 ID=1:IE=1:IG=1:IH=2
115 II=2:IS=0:JS=0:LS=0
120 GOSUB 1200
130 GOSUB 400:GOTO 130
199 REM GET NEXT CHAR IN C$ **********
200 IF IS>=LS THEN 205
201 IS=IS+1:C$=MID$(S$(JS),IS,1):IF ID=1 THEN PRINTC$;
202 RETURN
205 IF II=2 THEN GOSUB 350:RETURN
206 IF ID=1 THEN PRINT
207 IS=0:LS=0:IF JS>=NS THEN C$="":RETURN
208 JS=JS+1:LS=LEN(S$(JS)):GOTO 201
209 REM DECOMPOSE C$ INTO C%, C1% **********
210 GOSUB 200
215 IF C$="" THEN C%=0:C1%=0:RETURN
217 C%=ASC(C$):C1%=C% AND V7%:RETURN
229 REM HANDLE NOTE OR DURATION IN C$ **********
230 GOSUB 215
232 I%=PEEK(KC%+C1%):IF I%>V7% THEN ON IE GOTO 250,260
235 IF C%<>C1% THEN I%=I%+KS%
240 IF IE=2 THEN POKE L,(I% OR VC%):L=L+1:RETURN
245 POKE QQ%,I%:RETURN
249 REM DURATION: IMM, STO
250 I%=ND%(I% AND V7%):IF C%<>C1% THEN I%=I%+I%/2
255 POKE DL%,I%:I%=0:RETURN
260 I%=I% AND V7%:IF I%>6 THEN I%=I%+10
265 IF C%<>C1% THEN I%=I%+8
270 POKE L,(I% OR 64):L=L+1:RETURN
299 REM PLAY A NOTE AND MAYBE WAIT
300 Z=USR(X):QQ%=QL%:IF IG=1 THEN RETURN
305 GET Z$:IF Z$="" THEN ON IH GOTO 305,315
310 IF Z$="=" THEN IH=3-IH
315 RETURN
319 REM GET TEMPO INTO I% **********
320 GOSUB 210:I%=(C1%-64)*10:IF C%<>C1% THEN I%=I%+5
324 IF I%<10 THEN I%=10
326 IF I%>255 THEN I%=255
328 RETURN
339 REM SWAP BOARDS **********
340 KB=K2+K4-KB:KS%=S2+S4-KS%:KC%=KB-32
342 IF ID=1 THEN PRINT"<";MID$("24",(KB=K2)+2,1);">";
344 RETURN
349 REM GET CHAR INTO C$
350 GET C$:IF C$="" THEN 350
355 RETURN
```

```
400 REM PLAY IMMEDIATE NOTES **********
405 GOSUB 200:IF C$="." THEN RETURN
410 IF C$="/" THEN 450
415 IF C$="[" THEN 470
420 IF C$="]" THEN IS=IX:JS=JX:LS=LX:GOSUB 490:GOTO 400
425 IF C$=">" THEN 510
430 IF C$="*" THEN GOSUB 320:POKE TL,IZ:GOTO 400
435 IF C$="@" THEN GOSUB 340:GOTO 400
440 GOSUB 230:IF IZ THEN GOSUB 300
442 GOTO 400
449 REM CHORD, IM
450 GOSUB 200:IF C$="/" THEN GOSUB 300:GOTO 400
455 GOSUB 230:IF QQZ<QHZ THEN QQZ=QQZ+2
460 GOTO 450
469 REM BEGIN PHRASE, IM
470 GOSUB 210:PN=CZ-48
475 PIZ(PN)=IS:PJZ(PN)=JS
480 GOSUB 200:IF C$<>"]" THEN 480
485 GOSUB 490:GOTO 400
490 IF ID=1 THEN PRINT
492 RETURN
509 REM USE PHRASE, IM
510 IX=IS:JX=JS:LX=LS:GOSUB 210:PN=CZ-48
515 IS=PIZ(PN):JS=PJZ(PN):LS=LEN(S$(JS)):GOSUB 490:GOTO 400
600 REM STORE MUSIC IN (L) **********
605 GOSUB 200:IF C$="" THEN RETURN
610 IF C$="/" THEN 650
615 IF C$="[" THEN 670
620 IF C$="]" THEN POKE L,253:L=L+1:GOTO 600
625 IF C$=">" THEN 680
630 IF C$="*" THEN GOSUB 320:POKE L,249:POKE L+1,IZ:L=L+2:GOTO 600
635 IF C$="@" THEN GOSUB 340:GOTO600
640 GOSUB 230:GOTO 600
649 REM CHORD
650 VCZ=0:GOSUB 200
655 GOSUB 215:GOSUB 200:IF C$<>"/" THEN GOSUB 232:GOTO 655
660 VCZ=V8Z:GOSUB 232:GOTO 600
669 REM BEGIN PHRASE
670 GOSUB 210:PN=CZ-48:PIZ(PN)=IS
675 POKE L,254:L=L+1:POKE L,PN:L=L+1:GOTO 600
679 REM PLAY PHRASE
680 GOSUB 210:PN=CZ-48:PIZ(PN)=IS
685 POKE L,252:L=L+1:POKE L,PN:L=L+1:GOTO 600
1200 REM SETUP POINTERS
1202 LH=28:LP=256*LH
1205 POKE 59459,255
1210 K4=LP+3*256+54:K2=K4+64
1215 S2=-24:S4=1
1220 KB=K2:KSZ=S2
1225 Q1=31:Q2=LH+1
1230 SN=LP+512+24
1235 WT=LP
1240 PL=LP+256+12:QL=LP+256+23
1245 QLZ=QL:QHZ=QLZ+6:QQZ=QLZ
1255 DLZ=PL+8
1260 TL=LP+256+21
1265 POKE 1,Q1:POKE 2,Q2
1270 DU=128:POKE DLZ,DU/8
1275 V7Z=127:V8Z=128
1280 NDZ(0)=DU
1282 FOR I=1 TO 6
1284 :NDZ(I)=(NDZ(I-1)+1)/2
1286 NEXT I
1290 FOR I=7 TO 9
1292 :NDZ(I)=(NDZ(I-6)+1)/3
1294 NEXT I
1299 GOTO 340
```

```
1600 REM LIST SELECTIONS UNTIL #J
1610 I=0:RESTORE
1620 I=I+1:READ S$
1630 IF S$="END OF TUNES" THEN I=-1:PRINT:PRINTS$:RETURN
1635 PRINT I;:IF J=0 THEN PRINT S$
1640 IF I=J THEN RETURN
1650 READ S$:IF S$<>"END" THEN 1650
1660 GOTO 1620
2000 REM PLAY TUNES
2020 PRINT"ENTER TUNE NUMBER (0 TO LIST THEM)"
2025 INPUT J:IF(J=-1)AND(IE=2)THEN 2220
2030 GOSUB 1600:IF I<>J THEN 2000
2040 PRINT:PRINT"ENTER 1 TO PLAY AS IS, 2 TO STORE FIRST"
2045 INPUT IE:IF IE=2 THEN ID=2:GOTO 2100
2060 PRINT:PRINT"ENTER 1 TO DISPLAY NOTE INPUT, 2 NOT TO"
2065 INPUT ID:IF(ID<>1)AND(ID<>2)THEN 2060
2070 PRINT:PRINT"ENTER 1 FOR CONTINUOUS PLAY, 2 FOR DEBUG"
2075 INPUT IG:IF(IG<>1)AND(IG<>2)THEN 2070
2080 IH=3-IG
2100 REM LOAD TUNE *********
2105 NS=-1:GOSUB 1200
2110 NS=NS+1:READ S$(NS):IF S$(NS)<>"END" THEN 2110
2120 NS=NS-1:IS=0:LS=0:JS=-1
2130 L0=32768:L=L0
2140 IF IE=2 THEN PRINTCHR$(147);:VC%=V8%
2150 ON IE GOSUB 400,600
2160 IF IE=1 THEN 2000
2200 REM PLAY THE STORED TUNE
2210 POKE L,255
2220 GOSUB 2300:FOR I=1 TO 20:PRINT"        ";:NEXT I:GOSUB 2300
2230 PRINT"PRESS ANY KEY TO PLAY";:GOSUB 350
2250 POKE 1,32:POKE 2,LH+2
2260 Z=USR(L0/256)
2270 GOSUB 2300:GOTO 2000
2300 PRINTCHR$(19);:FOR I=1 TO 20:PRINT:NEXT I:RETURN
4000 REM LIST TUNES
4010 OPEN4,4,1:RESTORE
4020 READ S$:PRINT#4,"      ";
4030 FOR I=1 TO LEN(S$)
4040 :C=ASC(MID$(S$,I,1)):C1=C AND 127
4050 :IF C<>C1 THEN PRINT#4,"↑";
4060 :PRINT#4,CHR$(C1);:PRINTCHR$(C);
4070 NEXT I
4080 PRINT#4:PRINT
4090 GOTO 4020
```

```
 5000 DATA"MOONLIGHT SERENADE"
 5005 DATA"*R"
 5010 DATA"[12/BJE/SG8SRT2/NQE/DG8DRT"
 5012 DATA"2/NJ:/8DRT2/HJ:/8DRT"
 5014 DATA"2/SJET//SJRY/S/SJRY/"
 5016 DATA"/GTU/RE8S/YU//UO/"
 5018 DATA"2/GTU/EW8S/YU//UO/"
 5020 DATA"2/GRU/8WUO2/EU/8WUO"
 5022 DATA"2/DWTP//JWT//ND//DJY/"
 5024 DATA"2/MWTO//MHJR//BS//BSJT/"
 5026 DATA"2/NJEU//NHJL//NF//NJ:/]"
 5028 DATA"[22/SJRU/U/RYO/3XO]"
 5030 DATA"2W>1>2
 5032 DATA"2/BWTU/8 OP2/SJY$/W>1>2"
 5034 DATA"2/BTUO/S/GJEY//JT/"
 5040 DATA"3/MDTU/UUH2J/JY/"
 5042 DATA"3/HWTO/O2O/ON//NWTU/
 5044 DATA"2/FQEU/V7/BHWR/TY"
 5046 DATA"2/DWYU//WY/D/DWTU/
 5048 DATA"3/GEUI/I2I/FEY/R"
 5050 DATA"3/QET$/$2$/FERY/P"
 5052 DATA"7/JLEO/OO2/DJWT/C"
 5054 DATA"7/NETU/UU2/SFQW/W"
 5056 DATA">1*U>2*X"
 5058 DATA"2/BWTU/3/FJLY//DT/"
 5060 DATA"*Z1/GEUO/"
 5099 DATA"END"
 5200 DATA"TEST SCALE # 1"
 5205 DATA"*Z"
 5210 DATA"[1/WRYO//ETUP//RYI$//TUOZ//YIP'//UO$&//!#Z\//O$'(/]"
 5215 DATA"2>13>14>15>16>1*E>1*C>1>1*B>1>1>1>1"
 5299 DATA"END"
 5400 DATA"TEST SCALE # 2"
 5405 DATA"*J@"
 5410 DATA"[1XSCDVBGNHMJ,;LE#RTZY'U&IO"
 5412 DATA"SCDVBGNHMJ,;LE#RTZY'U&IO"
 5414 DATA"OI&U'YZTR#EL;,JMHNGBVDCSX"
 5416 DATA"I&U'YZTR#EL;,JMHNGBVDCSX]"
 5420 DATA"3>14>15>16>1>1*E>1>1>1*C>1>1>1>1>1>1*A>1>1>1>1>1>1>1>1"
 5449 DATA"END"
 5450 DATA"HAPPY BIRTHDAY"
 5455 DATA"*J@3XX2CXB1V3XX2CXN1B"
 5460 DATA"3XX2;M3BB2V1C3JJ2MBN1B "
 5465 DATA"2;?;0: "
 5499 DATA"END"
 5750 DATA"IF MY FRIENDS COULD SEE ME NOW!"
 5752 DATA"*G@[11MNV2M1NV2MNVM1NOV"
 5754 DATA"2;?;M?;M?;M1?;"
 5756 DATA"2,;?1::::2R#E1;2M,;1????2RE,1N"
 5758 DATA"2MNVM1NV2MNVM1;?"
 5760 DATA"2REWE1WM2?;ME1#1R"
 5762 DATA"2#R1TT2T1RE2?3LL2?REL?"
 5764 DATA"1RR2R1E0;1 ]"
 5766 DATA">1*D>1*B>1"
 5799 DATA"END"
 5800 DATA"JAWS"
 5805 DATA"*J@[1XSXSXSXS]"
 5810 DATA"0>11>12>13>14>15>15>1"
 5820 DATA"*Z0/;LY'/"
 5849 DATA"END"
 9999 DATA "END OF TUNES"
READY.
```

```
MOONLIGHT SERENADE
*R
[12/BJE/SG8SRT2/↑NQE/DG8DRT
2/N↑J:/8DRT2/H↑J:/8DRT
2/SJ↑E↑T//S↑JRY/S/S↑JR↑Y/
/GTU/RE8S/Y↑U//UO/
2/GTU/EW8S/Y↑U//UO/
2/GRU/8W↑UO2/↑EU/8W↑UO
2/DW↑TP//JW↑T//N↑D//D↑JY/
2/MWTO//MH↑JR//B↑S//BSJT/
2/N↑JE↑U//NH↑J↑L//NF//N↑J:/]
[22/S↑JRU/U/R↑YO/3XO]
2↑W>1>2
2/BWTU/8 OP2/S↑J↑Y$/↑W>1>2
2/BTUO/S/G↑J↑EY//JT/
3/↑MDTU/UUH2J/↑JY/
3/H↑WTO/O2O/ON//N↑WT↑U/
2/FQEU/V7/BH↑WR/TY
2/↑DWYU//WY/D/DW↑T↑U/
3/GEUI/I2I/FE↑Y/R
3/QET$/$2$/FER↑Y/P
7/JL↑EO/OO2/DJW↑T/C
7/NET↑U/↑U↑U2/SFQ↑W/↑W
>1*U>2*X
2/BWTU/3/FJLY//DT/
*Z↑1/GEUO/
END
TEST SCALE # 1
*Z
[1/WRYO//E↑TUP//R↑YI$//TUO%//YIP'//U↑O$&//!↑#↑%\//O$'(/]
2>13>14>15>16>1*E>1*C>1>1*B>1>1>1>1
END
TEST SCALE # 2
*J@
[1↑X↑S↑C↑D↑V↑B↑G↑N↑H↑M↑J↑,↑;↑L↑E↑#↑R↑T↑%↑Y↑'↑U↑&↑I↑O
SCDVBGNHMJ,;LE#RT%Y'U&IO
OI&U'Y%TR#EL;,JMHNGBVDCSX
↑I↑&↑U↑'↑Y↑%↑T↑R↑#↑E↑L↑;↑,↑J↑M↑H↑N↑G↑B↑V↑D↑C↑S↑X]
3>14>15>16>1>1*E>1>1>1*C>1>1>1>1>1>1*A>1>1>1>1>1>1>1>1
END
HAPPY BIRTHDAY
*J@3XX2CXB1V3XX2CXN1B
3XX2;M3BB2V1C3JJ2MBN1B
2;?;0:
END
IF MY FRIENDS COULD SEE ME NOW!
*G@[11MNV2M1NV2MNVM1NOV
2;?;M?;M?;M1?;
2,;?1::::2R#E1;2M,;1????2RE,1N
2MNVM1NV2MNVM1;?
2REWE1WM2?;ME1#↑1R
2#R1TT2T1RE2?3LL2?REL?
1RR2R1E0;1 ]
>1*D>1*B>1
END
JAWS
*J@[1↑X↑S↑X↑S↑X↑S↑X↑S]
0>11>12>13>14>15>15>1
*Z0/;LY'/
END
END OF TUNES
```

```
1 REM PETUNIA, VERSION 5
2 REM BY MARTIN J. COHEN, COPYRIGHT 1978 BY HUH ELECTRONICS
5 REM PROGRAM TO LOAD PETUNIA SUBROUTINE INTO MEMORY
9 CL$=CHR$(147)
11 PRINT CL$;
12 PRINT "MEANING, LOC, BYTES (HIGH, LOW)"
15 LH=28:L=256*LH:REM START OF STUFF/256
17 POKE 135,LH
18 DC=1
20 IF DC THEN READ V:GOTO 45
25 READ V$:V1=ASC(V$)-48:IF V1=-3 THEN V=VAL(V$):GOTO 45
30 IF V1>9 THEN V1=V1-7
35 V2=ASC(RIGHT$(V$,1))-48:IF V2>9 THEN V2=V2-7
40 V=16*V1+V2
45 IF V=-1 THEN GOSUB 600:V$="DONE":GOSUB 1020:GOTO 70
50 IF V=-2 THEN GOSUB 1000:GOTO 20
55 IF V=-3 THEN DC=1-DC:GOTO 20
60 IF V>255 THEN V=V-256+LH
65 POKE L,V:L=L+1:GOTO 20
70 PRINT:PRINT"PETUNIA SUBROUTINE IS NOW LOADED"
75 PRINT"AND PROTECTED."
100 END:REM 7934 (30,254)
115 DATA -2,"NOTE TABLE":REM 7168 (28,0)
120 DATA 51,52,53,54,54,55,56,57,57,58,58
130 DATA 59,59,59,60,60,60,60,60,60,60,60,60,60,60,60,60
140 DATA 59,59,59,59,59,58,58,58,58,58,58
150 DATA 57,57,57,57,57,57,57,57,57,57,58,58,58,58,58
160 DATA 59,59,59,59,60,60,60,61,61,61,61,62,62,62,62
170 DATA 63,63,63,63,63,63,63,63,63,63,63,63,62,62,62
180 DATA 61,61,60,60,59,59,58,57,56,56,55,54,53,52,51,50
190 DATA 49,48,47,46,45,44,43,42,41,40,39,38,37,36,35,34,33
200 DATA 33,32,31,31,30,30,29,29,29,29,28,28,28,28
210 DATA 29,29,29,29,29,30,30,31,31,32,32,33,33,34,35,35,36,36
220 DATA 37,38,38,39,40,40,41,41,41,42,42,43,43,43,43,43,43,43,43
230 DATA 42,42,42,41,41,40,39,39,38,37,36,35,34,33,32,31
240 DATA 29,28,27,25,24,23,21,20,19,17,16,15,13,12,11,9,8,7,6,5,4
250 DATA 3,3,2,1,1,0,0,0,0,0,0,0,0,0,1,1,1,2,3,4,5,6,7,8,9
260 DATA 11,12,13,15,16,18,19,21,22,24,26,27,29,31,32,34
270 DATA 35,37,39,40,42,43,44,46,47,48,49
280 DATA -2,"DATA AREA":REM 7424 (29,0)
290 DATA 0,0,256,0,0,256,0,0,256,0,0,256,0,0,0,0,0,0,0,0,0,82,0
291 DATA -2,"NOTE PROGRAM DATA AREA":REM 7447 (29,23)
292 DATA 0,0,0,0,0,0,0,0,-3
293 DATA -2,"NOTE PROGRAM":REM 7455 (29,31)
294 DATA 18,A2,08,BD,15,G1,2A,A8
295 DATA B9,D1,G2,9D,0A,G1,B9,D0,G2,9D,0B,G1
296 DATA A9,00,9D,15,G1,CA,CA,D0,E6
298 DATA -3
299 DATA -2,"MUSIC PROGRAM":REM 7484 (29,60)
300 DATA 162,21,189,0,257,149,53,202,208,248
310 DATA 120,160,0,166,74,24,177,54,113,57,113,60
320 DATA 113,63,141,65,232,165,53,101,65,133,53,165
330 DATA 54,101,66,133,54,165,56,101,67,133,56,165,57
340 DATA 101,68,133,57,165,59,101,69,133,59,165,60,101
350 DATA 70,133,60,165,62,101,71,133,62,165,63,101,72
360 DATA 133,63,202,208,8,198,73,240,12,166,74
370 DATA 208,185,208,0,208,0,208,0,208,177,88,96
```

```
400 DATA -2,"PLAY A SERIES OF NOTES":REM 7580 (29,156)
405 DATA -3,EA,EA,EA,EA
410 DATA 20,E0,G1,A0,00,20,F0,G1,C9,FA
415 DATA 90,01,60,C9,F9,D0,09,20,F0,G1
420 DATA 8D,15,G1,4C,A5,G1,29,7F,C9,40
425 DATA 90,0C,29,3F,AA,BD,00,G2,8D,14,G1
430 DATA 4C,A5,G1,99,17,G1,C0,06,F0,02
435 DATA C8,C8,20,F8,G1,10,CB,20,1F,G1,4C,A3,G1
440 DATA 20,A7,D0,A6,B4,CA,8E,FA,G1
445 DATA A9,FF,8D,F9,G1,60,00
450 DATA EE,F9,G1,D0,03,EE,FA,G1
455 DATA AD,00,00,60,00,00,00,00
459 DATA -3
460 DATA -2,"SERIES STORAGE":REM 7680 (30,0)
465 DATA 128,64,32,16,8,4,2,1
470 DATA 192,96,48,24,12,6,3,1
475 DATA 43,21,11,5,3,1,0,0
480 DATA 00,00,00,00,00,00,00,00
500 DATA -2,"MUSIC":REM 7712 (30,32)
505 DATA -3,20,E0,G1,A2,14,A9,00
510 DATA 9D,47,G2,CA,D0,FA
513 DATA 20,A3,G1,C9,FF,D0,01,60
516 DATA C9,FE,F0,27,C9,FC,F0,43
520 DATA C9,FD,F0,67,4C,2D,G2
522 DATA 00,00,00,00
524 DATA -2,"GRPLOC"
526 DATA -3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
530 DATA -2,"STRTGP"
532 DATA -3,20,C0,G2,30,0D,AA,AD,F9,G1
534 DATA 9D,48,G2,AD,FA,G1,9D,49,G2
536 DATA 20,F0,G1,C9,FD,D0,F9,4C,2D,G2,00,00,00,00
540 DATA 20,C0,G2,30,A8,AA,AD,F9,G1,8D,B8,G2,AD,FA,G1,8D,B9,G2
542 DATA BD,49,G2,F0,09,8D,FA,G1,BD,48,G2,8D,F9,G1,4C,2D,G2,00,00,00,00,00
544 DATA AD,B8,G2,8D,F9,G1,AD,B9,G2
546 DATA 8D,FA,G1,4C,2D,G2,00
548 DATA 00,00,00,00,00,00,00,00
552 DATA 20,F0,G1,10,01,60
554 DATA C9,0A,90,03,A9,FF,60,0A,60,00
570 DATA -2,"STANDARD NOTE TABLE":REM 7888 (30,208)
572 DATA -3,0,0
574 DATA 1,233,2,6,2,37,2,69,2,104,2,140
576 DATA 2,179,2,220,3,8,3,54,3,103,3,154
578 DATA 3,209,4,11,4,73,4,138,4,207,5,25
580 DATA 5,102,5,184,6,15,6,108,6,205,7,53
582 DATA 7,163,8,23,8,146,9,21,9,159,10,49
585 DATA 10,204,11,113,12,31,12,215,13,155,14,106
590 DATA 15,69,16,46,17,36,18,41,19,62,20,98
593 DATA 21,153,22,226,24,62,25,175,27,54,28,212
595 DATA 30,139,0,0
599 DATA -1
600 REM KEYBOARD TABLES
605 L4=L:L2=L+64
610 V$="4 OCTAVE KEYBOARD":REM 7990 (31,54)
615 GOSUB 1020:L=L+64
620 V$="2 OCTAVE KEYBOARD":REM 8054 (31,118)
625 GOSUB 1020:L=L+64
630 REM CLEAR KEYBOARD MEMORIES
635 FOR I=0 TO 63
640 :POKE L4+I,0
645 :POKE L2+I,0
650 NEXT I
```

```
655 REM GET NOTE POINTERS
660 DIM NP(9)
670 FOR J=0 TO 9:READ NP(J):NEXT J
675 DATA 0,1,3,5,6,8,10,12,13,15
680 REM FILL THE NOTE TABLES
685 W=ASC("Z")-32:POKE L2+W,24:REM 2 OCTAVE MIDDLE B
690 FOR I=0 TO 3
695 :READ C$
700 :DATA "ZXCVBNM,;?","ASDFGHJKL:","QWERTYUIOP","!"#$%'&\( )"
710 :K4=12*I:K2=12*INT(I/2) + (I AND 1)+ 24:REM LOCAL OCTAVE STARTS
715 :FOR J=INT((4-I)/4) TO 9-INT(I/3)
720 ::W=(ASC(MID$(C$,J+1,1)) AND 127)-32
725 ::POKE L4+W,K4+NP(J)
730 ::POKE L2+W,K2+NP(J)
735 :NEXT J
740 NEXT I
750 REM DIGITS ARE SPECIAL
755 FOR I=0 TO 9
760 :I1=I+128:I2=I+ASC("0")-32
765 :POKE L4+I2,I1
770 :POKE L2+I2,I1
775 NEXT I
780 REM END 8118 (31,182)
795 RETURN
1000 REM GIVE INFO
1010 READ V$
1020 L9=INT(L/256)
1030 L8=L - 256*L9
1040 PRINT V$;" AT"STR$(L);", (";STR$(L9);",";STR$(L8);")"
1070 RETURN
2000 REM HEX AND DEC DUMP
2010 PRINT:PRINT"ENTER LOW, HIGH LOC";:INPUT L0,L1:IF L1=0 THEN L1=L0+56
2020 L0=8*INT(L0/8):L1=8*INT(L1/8)+7
2030 HX$="0123456789ABCDEF"
2040 FOR L=L0 TO L1 STEP 8
2050 :D=INT(L/256):GOSUB 2400:H1$=H$
2060 :D=255 AND L:GOSUB 2400
2070 :PRINT:PRINT" ";H1$;H$;
2100 :FOR LL=L TO L+7
2110 ::D=PEEK(LL)
2120 ::GOSUB 2400
2130 ::PRINT"  ";H$;
2140 :NEXT LL
2150 :PRINT:PRINT RIGHT$("    "+STR$(L),5);
2160 :FOR LL=L TO L+7
2170 ::D=PEEK(LL)
2180 ::PRINT RIGHT$("   "+STR$(D),4);
2190 :NEXT LL
2200 :PRINT
2210 NEXT L
2220 GOTO 2000
2400 REM CONVERT D IN DEC TO H$ IN HEX
2410 D1=INT(D/16)
2420 H$=MID$(HX$,D1+1,1)
2430 D1=15 AND D
2440 H$=H$+MID$(HX$,D1+1,1)
2450 RETURN
```